FOAD python

17/12/2024

| * **Toute triche constatée sera sanctionnée par un 0/20. Assurez-vous de travailler individuellement et de ne pas copier le code d’autres sources.** * **Rendu pour le mardi 25/06/2024** |
| --- |

### Conseils Techniques :

**Approche progressive :**

* Développez une fonction à la fois
* Testez chaque fonction avant de passer à la suivante
* Commencez simple, ajoutez des fonctionnalités ensuite

**Débogage :**

* Utilisez des print() pour voir les valeurs
* Testez avec des cas simples
* Vérifiez chaque étape

**Organisation du code :**

* Commentez vos fonctions
* Utilisez des noms explicites
* Structurez votre code en sections logiques

# Exercice 1 : Mastermind

## Objectif :

Réaliser un jeu de Mastermind en Python. Ce jeu devra être fonctionnel dans un terminal. Des points bonus seront accordés pour ceux qui implémentent une interface graphique avec Pygame.

## Introduction au Jeu de Mastermind :

Le Mastermind est un jeu de déduction où un joueur doit deviner une combinaison secrète de couleurs (ou de chiffres). Voici les règles de base :

**La combinaison secrète :** Une série de 4 couleurs (ou chiffres) est choisie par le programme.

**Les tentatives :** Le joueur a un nombre limité de tentatives pour deviner la combinaison.

**Les indices** : Après chaque tentative, le programme fournit des indices :

* Un indicateur pour chaque couleur (ou chiffre) correcte à la bonne position.
* Un indicateur pour chaque couleur (ou chiffre) correcte mais à la mauvaise position.



## Consignes et Conseils pour le Développement :

### Initialisation du jeu :

* Générer une combinaison secrète de 4 couleurs (ou chiffres) aléatoires.
* Vous pouvez choisir parmi 6 couleurs (ou chiffres) possibles.

### Interaction avec le joueur :

* Demander au joueur de proposer une combinaison à chaque tour.
* Valider que la combinaison proposée est de la bonne longueur et ne contient que des couleurs (ou chiffres) valides.

### Fournir des indices :

* Comparer la combinaison proposée avec la combinaison secrète.
* Indiquer le nombre de couleurs (ou chiffres) corrects à la bonne position (avec un symbole, par exemple '\*').
* Indiquer le nombre de couleurs (ou chiffres) corrects mais à la mauvaise position (avec un autre symbole, par exemple '-').

### Gestion des tours :

* Limiter le nombre de tentatives à 10.
* Afficher un message de victoire si le joueur devine la combinaison avant la fin des tentatives.
* Afficher un message de défaite si le joueur n’a pas trouvé après 10 tentatives, et révéler la combinaison secrète.

### Bonus (Interface Graphique avec Pygame) :

* Vous pouvez créer une interface graphique avec Pygame pour rendre le jeu plus interactif et visuellement attrayant.
* Vous obtiendrez des points bonus si l’interface graphique est bien implémentée.

## Étapes de Développement Suggérées

### Étape 1 : Préparation (2 points)

1. Créez un nouveau fichier Python
2. Importez les modules nécessaires (pensez aux nombres aléatoires)
3. Définissez les constantes du jeu :
   * Quelles informations ne changeront pas pendant le jeu ?
   * Quelles valeurs doivent être facilement modifiables ?

🤔 **Réflexion** : Quelles sont les règles qui pourraient devenir des constantes ?

* La longueur de la combinaison
* Le nombre d'essais maximum
* Les valeurs possibles dans la combinaison

### Étape 2 : Génération de la Combinaison Secrète (2 points)

1. Créez une fonction qui va générer la combinaison secrète
2. Cette fonction doit :
   * Retourner une liste de la bonne longueur
   * Utiliser uniquement les valeurs autorisées
   * Être aléatoire à chaque partie

💡 **Indices** :

* Recherchez les fonctions Python pour générer des nombres aléatoires
* Comment créer une liste de N éléments aléatoires ?
* Pensez à tester votre fonction !

### Étape 3 : Interaction Joueur (2 points)

1. Créez une fonction pour récupérer la proposition du joueur
2. Cette fonction doit :
   * Demander au joueur de saisir sa combinaison
   * Vérifier que la saisie est valide
   * Convertir la saisie en liste de nombres

⚠️ **Points d'attention** :

* Que faire si le joueur entre autre chose que des chiffres ?
* Comment gérer une saisie trop courte ou trop longue ?
* Comment séparer les chiffres saisis ?

### Étape 4 : Vérification de la Proposition (2 points)

1. Créez une fonction qui compare la proposition avec la solution
2. Cette fonction doit :
   * Compter les chiffres bien placés
   * Compter les chiffres présents mais mal placés
   * Ne pas compter deux fois le même chiffre

🔍 **Méthode suggérée** :

1. Commencez par compter les bien placés
2. Pour les mal placés, attention aux doublons !
3. Testez avec des cas simples d'abord

### Étape 5 : Boucle de Jeu Principale (2 points)

1. Créez la fonction principale qui va :
   * Initialiser le jeu
   * Gérer les tours
   * Afficher les résultats
   * Détecter la fin de partie

📝 **Structure suggérée** :

1. Initialisation
2. Boucle principale (tant que pas gagné ET essais restants)
3. Gestion de fin de partie

### Étape 6 : Affichage et Interface Utilisateur (2 points)

1. Améliorez les messages affichés :
   * Instructions claires
   * Résultats lisibles
   * Messages de victoire/défaite

🎨 **Suggestions d'amélioration** :

* Comment rendre les résultats plus visuels ?
* Quelles informations sont utiles au joueur ?
* Comment rendre l'interface plus intuitive ?

### Bonus : Interface Graphique avec Pygame (4 points bonus)

Si vous avez terminé les étapes précédentes :

1. Installez Pygame
2. Créez une fenêtre de jeu
3. Affichez le plateau
4. Gérez les interactions souris
5. Ajoutez des animations

# Exercice 2 : Conway’s game of life

## Objectif:

**à regarder :** https://www.youtube.com/watch?v=S-W0NX97DB0

Le but du projet est de coder l’algorithme permettant de simuler le jeu de la vie.

Les règles du jeu sont les suivantes (source : https://fr.wikipedia.org/wiki/Jeu\_de\_la\_vie) :

| Le jeu se déroule sur une grille à deux dimensions (matrice) dont les cases sont appelées «cellules», par analogie avec les cellules vivantes.  Chaque cellule peut prendre deux états distincts : « vivante » ou « morte » (1 ou 0).  Une cellule possède huit voisins, qui sont les cellules adjacentes horizontalement, verticalement et diagonalement.  À chaque itération, l'état d’une cellule est entièrement déterminé par l’état de ses huit cellules voisines, selon les règles suivantes :   * une cellule morte possédant exactement trois cellules voisines vivantes devient vivante (elle naît) ; * une cellule vivante possédant deux ou trois cellules voisines vivantes le reste, sinon elle meurt ; |
| --- |

## Instructions :

Pré-requis :

* Une installation python
* Une installation de numpy

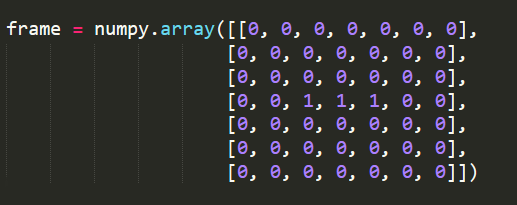
1. **Créez un répertoire de projet dans lequel vous créez un script : main.py**

Nous voulons modéliser une grille du jeu de la vie de taille 7x7 pour commencer.

On considérera dans la suite du projet que la grille est une matrice de dimension (7, 7) dont les éléments peuvent valoir 0 ou 1 (cellule = élément) :

* 0 si l'élément est mort
* 1 si l'élément est vivant

1. **Importez numpy à partir de votre script main.py et créez une variable frame tel que:**



*Pour savoir si une cellule va prendre vie ou mourir il faut :*

*connaître son état (1 ou 0) et connaître le nombre de voisins qu’elle possède.*

**Nous allons voir maintenant comment calculer le nombre de voisins d’une cellule.**

**Pour calculer le nombre de voisin vivant d’une cellule il suffit de sommer les valeurs de ses 8 cases voisines. Le résultat de cette somme est égale au nombre de voisins.**

**Problème : Comment faisons-nous pour les cases se situant sur les bords de la matrice ? Elles n’ont pas 8 cellules voisines !**

**Solution : Le zero padding !**

Nous allons entourer notre matrice avec des 0 pour créer une sorte de bordure artificielle.

Nous rajoutons donc :

* 2 lignes avec que des 0 (en haut et en bas).
* 2 colonnes avec que des 0 (droite et gauche)
* Le contenu de la grille du jeu de la vie dans la matrice avec bordures commence donc à la ligne 1, colonne 1.
* Le contenu de la grille du jeu de la vie dans la matrice avec bordure se termine donc à la ligne m-2 et à la colonne n-2 pour une grille de jeu de taille (m,n)

L'intérêt de rajouter des bordures (zero padding) est de nous permettre de calculer plus facilement le nombre de voisins sans avoir à se soucier de l’effet de bord lors des calculs.

En effet on peut parcourir la nouvelle matrice générée avec le padding

* de la ligne **1** à la ligne **m-2**
* de la colonne **1** à la colonne **n-2**.

Avec ce parcours sur cette matrice avec bordure, tous les éléments de la grille auront 8 voisins y compris ceux qui causaient problème.

1. Voici le squelette de l’algorithme avec du pseudo code:

